Fluid Tool - Product Description

William Scherlis, Carnegie Mellon University, Feb 28, 2005

What is it? The Fluid Tool provides the working developer with a means to detect potential race conditions in Java programs and assure their absence. The Tool is integrated into the widely adopted Eclipse development environment for Java. It can be used by individual developers, by software teams, and by acceptance evaluators.

Features: The Fluid Tool is distinguished from other source-code analysis tools in four ways: (1) It provides positive assurance, in the sense that there are no false negatives. (2) It addresses the hardest errors—the errors that defy conventional testing and inspection. (3) The tool enables developers to express critical design intent. (4) The tool does not give false positives due to the use of explicit design intent. Design intent is expressed as one-line JavaDoc-style annotations.

Benefits: The Fluid Tool provides static assurances for critical multi-threading properties that are difficult or impossible to assess using traditional testing, inspection, and runtime checking techniques. The analyses provide both "good news" and "bad news"—the tool provides assurance of absence of races (with respect to intent) as well as detection of race-triggering faults. As a developer evolves a system, the tool tracks consistency of model and code.

Successes: The Tool has been applied to at-scale Java production systems and components in aerospace, government, and top-10 software vendors. In every case, including widely used library code, it has found faults that can trigger race errors (not false positives). In several of these case studies, the Fluid team was able to rapidly locate faults that had eluded intense bug-finding effort.

Contexts in which it is best used: The Fluid Tool works most effectively on systems that are decomposed into subsystems—this allows for rapid realtime iteration of analysis. Code must build in the tool environment before it can be analyzed.

Compare with alternate known products or technologies. (1) Testing tools may not detect intermittent errors. (2) Inspection may not find triggering faults when errors are "non-local." (3) Model checking tools detects many errors, but there are difficulties in scale-up and composition. (4) Rule-based bug finding tools in general yield false positives because design intent is guessed. (5) Program verification systems are useful only at very small scale and for specialized requirements. "Compromise" approaches (e.g., ESC/Java) tend to require extensive program annotation. (7) Runtime monitoring cannot address most race errors, aliasing errors, etc.

What will a successful collaboration look like?

What will the technology provider do? Support closely coordinated collaboration for an initial period of two to four days. This provides *in situ* training in tool use and modeling.

What should the development team do? The NASA development team should appoint one or two liaison members. The team should assure the system can build in Eclipse.

How will the technology provider work together with the development team to ensure a successful collaboration? After the initial intensive period of collaboration, the Fluid team will provide some distance support for modeling, assurance, and enhancement to the tool.